CIT 591 Introduction to Software Development

Module 4: Solving Problems in an Object-Oriented Manner

Module Learning Objectives

- Interpret specifications for a project to figure out classes, methods
- Use the Nouns and Verbs approach to break up a project into classes
- Represent each action in the problem as classes interacting with each other via their methods

Module Glossary

- **Abstraction:** One of the key concepts of object-oriented languages like Java. Its main goal is to handle complexity by hiding unnecessary details (making them abstract) from the user.
- **`nextInt()`**: An inbuilt method in the Random class used to generate a random integer. When no arguments are specified it picks a random number from 0 to the maximum possible. Full details >
- **`random()`**: An inbuilt class in Java used for random number generation. Full details >

Module Resources

- **Textbook Readings:**
  - none
- **Websites:**
  - Class Random (Java Documentation)
  - Barbara Liskov's talk on "The Power of Abstraction"
    Note: We'll be seeing more of Barbara Liskov later in this course when we study the principle named after her!

Key Concepts & Examples

From a problem statement to code: Object-Oriented Programming and Designing for Class Interaction

Your programs will most frequently require classes to interact with other classes. This is hard to code on the fly, so it's easier to work from your problem statement or project specifications and consider how those specifications can turn into the classes and methods you will actually be writing.

It can help to break down a problem into its constituent classes and methods by applying the concept that a noun (player, league) is a class and a verb is a method (foul, hand out cards).

- Example used in the video: how to interpret a problem statement describing the rules for soccer players to be ejected from a game due to committing a foul.

Key to this approach (and programming in any object-oriented programming language) is **abstraction**. The main goal of abstraction is to handle complexity by hiding unnecessary details (making them abstract) from the user.

For example, consider that you have a Student class which has a takeCourses method. A fellow developer who wants to use your Student class while writing a School class should not have to worry about how the takeCourses method got implemented.